Debug - The 8088 assembler

To avoid a lot of confusion when you work with the debug program, the 8088 CPU instruction set and assembly programming, refer to the following list showing the commands and instructions of each one.

It is important for you to understand that:

Debug has a set of commands
The microprocessor has an instruction set
Assembly language programming has a set of mnemonic instructions and pseudo-operations.

Debug commands A C D E F G H I L M N O P Q R S T U W ?

8088 Instructions

The microprocessor is a digital circuit that only responses to a series of one's and zero's. Data is input to the CPU on the data pins marked D0 - D7. This is equal to one byte or 8 bits. If the first byte of data that enters the processor is an instruction, the processor will respond by performing the proper operation. In other words, the CPU has a set of 8 bit or one byte codes that it response's to, that will cause an operation to take place inside the registers or on the buses of the system board. These codes are called instructions and all of the codes combined are referred to has the instruction set.
The following examples of several 8088 instructions will help you understand. These are real instructions that preform the specified operation.

Byte D7--D0----------operation performed

1. 10111000 ----moves the next two bytes into AX
2. 00000101 ----LSB of the number (goes in al)
3. 00000000 ----MSB of the number (goes in ah)
4. 00000001 ----adds BX to AX- result goes in BX
5. 10001001 ----moves the contents of BX to the
6. 00011110 ----memory location pointed to by
7. 00100000 ----the next two bytes.
8. 00000001

There are 256 possible combinations of an 8 bit binary number that can be used as an instruction. It is virtually impossible for a human being to remember and work with all of these one's and zero's of the 8088 instruction set. It has become necessary for us to represent these binary numbers with the base 16 number system.(Hex) The Hex representation of binary does make it a little easier to work with. For example, the program listed above in machine code (1's and 0's) can be represented with the Hex digits for the instructions as shown below.

Byte # D7-D0----------operation performed

1. B8 ----------moves the next two bytes into AX
2. 05 ----------LSB of the number (goes in al)
3. 00 ----------MSB of the number (goes in ah)
4. 01 ----------adds BX to AX- result goes in BX
5. 89 ----------moves the contents of BX to the
6. 1E ----------memory location pointed to by
7. 20 ----------the next two bytes.
8. 01

This kind of programming is called machine code because the instructions are entered using binary or Hex data which, is the real data the CPU is responding to. The complete set of machine instructions will not be listed here because it would serve no purpose at this time. If you find a need for this information you must refer to the INTEL 8088 technical manual.

Assembly language

Programming in the early days of computing had to be written using the machine code of the microprocessor and as a result required specialized people who had this ability. The use of computers was restricted to the companies and corporations with the finances to support such a specialized operation. It didn't take long for the programmers to write the code to do most of the hard work in binary. One of the most important programs written was the Assembler, which was a conversion type program that allowed the use of English words to represent the instruction set of the CPU, that was compiled into machine code after they were written. One such program is Debug.com that is supplied by the MicroSoft corp. in most versions of DOS.

Debug, which is a small but powerful assembler, converts English sounding words called mnemonics into the machine code. All instructions for the 8088 have a mnemonic word to represent them in debug. For example, in the program above, the code can be written in an assembler language that is much easier to write and understand.

The mnemonic form of that program is shown below.

mnemonic form----------machine code

MOV AX,0005----------B80500
ADD BX,AX----------01C3
MOV [120],AX----------891E2001

Debug Command Description

A- Assembles 8086/8088 mnemonics.
C- Compares two areas of memory.

D- Displays the contents of memory locations.
E- Enters data into specified memory locations.
F- Fills memory locations with specified values.
G- Runs an executable program from within debug.
H- Performs Hexadecimal math on two Hex numbers.
i- reads one byte of data from the specified port.
L- loads the contents of a disk file or sectors into memory.
M- moves or copies the contents of memory locations.
N- Names a file for the L or w command. (load or write)
O- outputs one byte of data to the specified port.
P- Executes a loop, a repeated string instruction, a software interrupt, or a subroutine.
Q- Quits a debug session.
R- Displays or modifies the contents of the cpu registers.
S- Searches the contents of memory for the specified string of data.
T- Executes one instruction at a time and dumps the contents of all the registers to the screen, then displays the next instruction to be executed.
U- Disassembles machine code into the mnemonic form of the program.
W- Writes the data in memory to the disk. The name given the file is the name entered with the **n** (name) command. The file size is given by setting the BX:CX register to the number of bytes to be saved.
?- Display a help screen on some versions of debug.

A (assemble) Assembles 8088/8086 mnemonics into memory. This command is used to create executable machine code from assembly-language statements. All numeric values are in hexadecimal form and can only have from 1 to 4 digits.

SYNTAX - A [address]

PARAMETER
address - the location where you will store the assembly-language instruction. The address must be in Hexadecimal form and within the range of 0100-FFFF. If you don't specify an address, debug will default to the last address used.

An assembly-language instruction is made up of a mnemonic instruction word followed by none/one or two operands. The format of an instruction is shown below.

MNEMONIC 1st parameter, 2nd parameter

----------(destination) (source)
op-code 1st operand, 2nd operand

MOV AX , BX

The op-code part of the instruction (MOV) tells the processor what to do.
The 1st parameter (AX) is the 1st location of data that the operation will be performed on.
The 2nd parameter (BX) is the second location of data that the operation will be performed on.
These parameters are called operands.
The 1st operand is the destination of the data, the second operand is the source of data.
In other words data moves from the second operand to the first operand.
The above move instruction above will move the contents of the BX register into the AX register.
More examples:

MOV AX,45 load the hex number 45 into the AX register
MOV AX,[120] load the contents of location 120 into the AX register
ADD AX,[500] add contents of location 500 to AX ,save the results into AX
Note: You must start assembly of a COM program at offset 100.

C (compare) Compares two blocks of memory.

SYNTAX C range address

PARAMETERS
range - specifies the starting and ending addresses or the start address and length, of the first block of memory to compare. address - specifies the starting address of the second block of memory you want to compare. Example: C 100 220 500 compares the block of memory at offset 100 - 220 with the block starting at offset 500. C 100 L 120 500 compares the block of memory at offset 100 for a length of 120 to the block of memory starting at 500.

D (dump) Displays or dumps the contents of memory onto the display.

SYNTAX D [range]

PARAMETER
range - specifies the starting and ending address or the starting address and the length, of memory to be displayed. If you don't specify a range debug will display 128 bytes starting at the end of the last address dumped.

Note: When you use the D command, debug displays memory contents in two portions: a hexadecimal (each byte value is shown in hex form) and an ASCII portion (each byte shown as an ascii character).

seg:offset---------- Hexadecimal portion ---------- ASCII
0A1F:0100 - 34 33 00 F1 00 00 6C 1B-24 44 44 A9 E1 12 11 41 .....
0A1F:0110 - EF 56 A8 4B 12 30 E5 31-63 35 28 94 27 38 00 28 .....
0A1F:0120 - 12 00 00 17 4D FF FA 1F-1B 24 00 00 00 00 00 00 .....
Example:

D 100 10F

Will display the following,
0A1F:0100 -- 41 42 43 44 45 46 47 48-1B 24 00 00 00 00 00 00 ABCDEFG.....

E (enter)
Enters data into the memory location specified. Data can be entered in hexadecimal or ascii format. Data at the specified location will be lost and replaced with the new data.

SYNTAX E address [list]

PARAMETER
address - specifies the first location in memory for the entered data.

list - specifies the data that you want to enter into successive bytes of memory.

Using the address parameter

If you specify a value for the address without specifying the LIST parameter, debug displays the address and it's contents and waits for you to enter you new data. At this point you can do one of the following.

1. Replace the byte value with a new value in hex. If the value you enter is not valid hexadecimal, debug will inform you of that.
2. Advance to the next byte in memory. To do this you press the space bar, and to change the value at the new location, you just type a new value. If you press the space bar more than eight times, debug starts a new line.
3. Return to the preceding byte. To do this you press the hyphen key. Each time you press the hyphen key, debug will go back in memory one byte.
4. Exit from the E command. To do this you just press the enter key.

Using the LIST parameter

If you specify values for the list parameter, the old value will be replaced with the new value. LIST values can be hexadecimal or string values. To enter string values, just enclose the string within single or double quotes.

Example:

If you type,

E 100

debug displays the contents of offset 100

0A1F:0100 41._

To change this value you just type in the new value,

0A1F:0100 41.35

In this example, the value 41 at offset 100 was changed to 35.

You can enter consecutive byte values with one E command. Instead of pressing the enter key after you change a value, just press the space bar and debug will display the next location in memory, at which time you can enter a new value. Repeat this process as many times as you need to.

If you enter,

E 100

debug returns the value at offset 100

0A1F:0100 35._

if you press the space bar three times then debug shows the next three offsets.

0A1F:0100 35._ 42._ 43._ 44._

To change the byte at offset 100 you can hit the hyphen key three times and debug will go back to offset 100.
0A1F:0102 43._
0A1F:0101 42._
0A1F:0100 35._

Press enter to get out of the E command.

To enter data into a location using a String of text, enclose the string in quotes.

E 100 "This is a string of text."

This command will enter the string of ascii characters into the 25 memory locations starting at offset 100.
(one location for each character and spaces)

F (fill)

Fills the addresses in the specified memory locations with the value you specify. You can specify the data in hexadecimal or ascii values.

SYNTAX F range list

PARAMETERS
range - Specifies the starting and ending address, or the starting address and the length of the memory area you want to fill.

list - Specifies the data you want to fill with. Data can be hexadecimal or ascii form. The ascii form is a string of text enclosed in quotes.

Using the range parameter

If range contains more bytes than the number of values in LIST, debug assigns the values in LIST repeatedly until all bytes in range are filled. If any of the memory in the range specified is bad or doesn't exist, debug will return an error message to that affect.

Using the list parameter

If list contains more values than the number of bytes specified in range, debug will ignore the extra values.

G (go)

Executes the program that is in memory. The go command uses the IP register as a pointer to the next instruction that will be executed.

SYNTAX G [=address] [breakpoint]

PARAMETER
address - Specifies the address that program execution will begin at.
If you do not specify an address, debug executes the instruction at the address in the CS:IP registers.
If the program ends with an INT 20 instruction, the IP register is reset back to offset 100.
If the program ends with an INT 3 instruction, the IP register will remain pointing to the next instruction after the last instruction executed.

breakpoint - Specifies from 1 to 10 breakpoints that can be entered with the Go command.

Using the address parameter

You must precede the address with an equal sign (=) to distinguish the starting address from the

breakpoint address.

Specifying breakpoint

The program stops at the first breakpoint that it encounters and dumps the contents of all registers, the status of the FLAGS and displays the last instruction that was executed.

Examples:

If you type,

G cs:200

debug will execute the program up to offset 200, then dump the contents of all the registers on the screen.

If you type,

G=cs:200

Debug will execute the program starting at offset 200 to the end of the program.

If you type,
G
Debug will use the CS:IP register to get the address for the next instruction to be executed. You should always look at the IP register before you enter g, to make sure that it is pointing to the next instruction that you want to execute. To look at the IP register, just enter rip (return), debug will display the current value in IP and prompt you for a new value if you desire. The IP register should always be at 0100 to run the entire program.
Be careful when ending the program with an INT 3 because all registers are preserved and are not reset to their initial values, IP will be pointing to the next address after the end of the program and not your program.
INT 20 will reset all the registers to zero and the IP to 0100 when the program ends.
See P and T commands

h (Hex)

Performs hexadecimal math on the two parameters you supply.

SYNTAX h value1 value2

PARAMETER

value1 - represents any hex number between the range of 0000 - FFFFh.

value2 - represents any hex number between the range of 0000 - FFFFh.

Debug first adds the two parameters you specify and then subtracts the second from the first. The results are displayed on one line. First the sum, then the difference.

If you type,

h FF 2C5

debug does the calculations and displays,

03C4 FE3A

03C4 is the sum and FE3A is the difference.

i (input)
inputs and displays a byte from the specified port

PARAMETER
port - specifies the port address. The port address can be an 8 or a 16 bit value. It is possible to have up to 65,535 port addresses.(FFFF)

See port

If you type

i 3bc

debug will display 1 byte of data from the parallel port.

L (Load) loads a file or the contents of the specified sector from the disk into memory.

SYNTAX L address

L address drive start number

PARAMETER
address - Specifies the memory location where you want to load the file or sector contents. If you don't specify an address, debug will use the address in the CS:IP registers

drive - specifies the drive that contains the disk that specific sectors are to be read. This is a number from 0 - 4.
0 = A
1 = B
2 = C
3 = D

start - specifies the hexadecimal number of the first sector whose contents you want to read.

number - specifies the hexadecimal number of sectors you want to load.

Using L with no parameters.

If you enter L without parameters, debug will re-load the file that was specified on the DEBUG command line into memory at offset CS:0100. Debug also sets the BX:CX registers to the number of bytes for the file size. If you didn't enter a file name on the command line debug will load the file that was specified by the n command.

Note:
The name of the file debug is using will be stored at offset 0080. To see the current name, use the d command to dump offset 80. Enter d 80 and debug displays the name of the current file. You should get in the habit of doing this so the file or program you are working on will be sure to get saved to the disk.

Using L with the address parameter

If you use L with just an address parameter, debug will load the data at the specified offset in memory.

Using L with all parameters
If you use all parameters, debug will load the specified sectors into the specified offset.

Loading an EXE program.
Debug ignores the address parameter for EXE files. If you specify an EXE file, debug relocates the file to the loading address specified in the header of the EXE file. The header itself is stripped off the EXE file before it is loaded into memory, so the file size on the disk will be different than the file size in memory. If you want to examine an EXE file, rename it with a different extension.

REN FILE1.EXE FILE1.TXT

Example: Suppose you start debug and at the prompt you type,

-n Filename.com

You can then enter L and debug will load the file with the name you specified from the disk.

Suppose you want to load the root directory from the disk in the A drive. The root directory starts at sector number 5 and is a total of 7 sectors in size.

You enter,

L 0100 0 5 7

debug loads in the 7 sector starting at sector 5 from the disk in the A drive and stores it at offset 0100.

m (move)
Copies the contents of a block of memory to another block of memory.

SYNTAX m range address
PARAMETER
range - specifies the starting and ending addresses, or the start and length of the memory area whose contents are to be copied.

address - specifies the starting address of the location where you want to copy the contents of range.

Example:

If you type the following command,

m 0100 110 500

Debug copies the block of data that starts at offset 100 and ends at offset 110, to offset 500 in memory.

Note:
You can use this command if you have to insert an instruction in the program that has already been written. Suppose you have to insert the instruction Mov ax,5600 in the program you are working on at offset 160.

Enter,

m 160 FFFF 1000

Then assemble the instruction at offset 160.

A 160

CS:0160 MOV AX,5600 (return)

To get the rest of the program back together (it was moved to offset 1000)
Enter,

m 1000 FFFF 163 (163 is the next offset

after MOV AX,5600)

The program should all be in order now.

n (name)
Specifies the name of a file that will be used by the L or the W command.

SYNTAX

n [drive:][path]\filename

PARAMETER
[drive:][path]filename - specifies the location and name of the file you want to work with.

Example:
Suppose you started debug with no filename on the command line, at the debug prompt you just enter the filename you want.

-n break.com

Then enter L and the file is loaded into memory.

Suppose you are working on a program and you want to work on a different program, just enter,

-n filename.ext

L (return)
and the new file is loaded into memory.

o (output)- sends one byte to the specified port.

SYNTAX o port byte

PARAMETER
port - specifies the output port address. Can be an 8 or a 16 bit port.

Byte - specifies the hexadecimal that will be sent to the specified port.

Example:

To send the letter A to the parallel printer, Just enter,

o 3bc 41

See port

p (proceed)
Executes a loop, a repeated string instruction, a software interrupt, or a subroutine, or traces through any other instruction.

SYNTAX p [=address] [number]

PARAMETER
address - specifies the address of the first instruction to be executed. If you don't specify an address, debug will use the address in the CS:IP registers. (see G command)

number - specifies the number of instruction to execute before returning control to debug. The default is one.

Note:
p is similar to g or the t command and is used for debugging a program by single stepping through the program. The p command however, will execute loops and INT instructions. The p command ***cannot*** be used to trace through the ROM bios program.

Q (quit) Exits debug and returns control to DOS.

SYNTAX Q

PARAMETERS none

NOTE:
Be sure to write your program or data to the disk before you enter Q, otherwise it will be lost.

R (register) displays or changes the contents of one or more of the micro-processor registers.

SYNTAX r [register-name]

r by itself will display all of the registers

rf displays the flag register

PARAMETER register-name - specifies the name of the register you want to display.

Using the r command
If you specify a register name, debug displays the 16 bit value of that register in hexadecimal form followed by the colon. If you want to change the value of the register, type the new value plus enter. If you don't want to change the value just press the enter key.

Valid register names

The following are valid register names.
AX BX CX DX
SP BP SI DI
ES SS DS CS
IP F

Using the F character (FLAG)
If you use the F character instead of a register name debug displays the current status of the flags register. Each flag has a two letter code to shown the condition of the flags. To set or clear the flags use the following list of two letter codes.

FLAG NAME----------SET----------CLEAR

Overflow----------ov----------nv
Direction----------dn----------up (increment)
Interrupt----------ei (enabled)------di (disabled)
Sign----------ng (neg)----------pl (positive)
Zero----------zr----------nz
Auxiliary carry----------ac----------na
Parity----------pe (even)---------po (odd)
Carry----------cy----------nc

Default settings for debug

When you start debug, the segments registers are set to what ever segment is free in your computer at the lowest possible memory location, the IP register is set to 0100, all flags are cleared and the remaining registers are cleared to zero except the SP register, which is set to FFFE.

Example:

To put the number 1234 into the AX register, enter

RAX (return)

you will see

AX 0000:

after the colon enter 1234 (return)

To see just the flags register enter,

rf (return)

you will see

NV UP DI NG NZ AC PE NC- _

at the hyphen enter the two letter code for the flag,you want to affect.

pl ei cy (return)

S (search) searches a range of addresses for a pattern of one or more bytes.

SYNTAX s range list

PARAMETER
range - specifies the start and end address of the range you want to search.

list - specifies the pattern of bytes or a string that you want to search for. Separate each byte value with a comma or a space and enclose the string in quotations marks.

Example:

To search a block of memory for the pattern of bytes, 41 55 66 located at offset 100 thur 250

Enter,

s 100 200 41 55 66 (return)

debug will display the address of all locations that contain this pattern of bytes.

To search the same block of memory for the string, Data Institute, Enter

s 100 250 "Data Institute"

debug will return all of the locations that contain that string.

Note:
You must have the exact bytes or the proper case letters for the search command. It is case sensitive.

t (trace) Executes one instruction at a time and displays the contents of all of the registers, the condition of the flags and the next instruction to be executed.

SYNTAX t [=address] [number]

PARAMETER
address - specifies the start address to start tracing. If you don't specify the start, debug will use the CS:IP registers for the next instruction.

number - specifies the number of instructions to be executed. Default number is one.

Note:
t is like the g and p command except it cannot go through a sub-routine or an INT instruction. I Would recommend using the p command for all single stepping and debugging.

However, t can be used to trace through the ROM bios program. If you want to unassemble the ROM bios program use t.

u (unassemble) - disassembles bytes and displays their source code listing in mnemonic form with their addresses and values.

SYNTAX u [range]

u (by itself u will unassemble 20 bytes)

PARAMETER
range - specifies the starting and ending addresses or the starting address and the length, of code

you want to unassemble.

Example:

To unassemble 16 bytes (10hex) starting at 0100

enter,
u 0100 L 10

debug displays,

CS:0100 MOV AX,5600
CS:0102 MOV DX,200
CS:0105 INT 21
CS:0107 MOV AH,7
CS:010B CMP AL,1B
CS:010D JZ 0111
CS:010F JMP 0107

You could have also entered,

u 100 110

and debug would display the same information.

w (write) Writes a file or a specific number of sectors to the specified disk.

SYNTAX w address drive start number (to write sectors to the disk)

w address (to write files to the disk)

PARAMETER
address - specifies the beginning memory address of the file you want to write to the disk file. If you don't specify, debug will start at CS:0100.

drive - specifies the drive that will be the destination disk.

0 = A
1 = B
2 = C
3 = D

start - specifies the start sector number in hexadecimal that you want to write to.
number - specifies the total number of sectors you want to write.

Saving a file to the disk

NAME

To write a file to the disk you must first, use the name command (n) to name the file. You must specify the path in the name command such as A:\advanced\break.com.

SIZE

After the file is named, you must specify the size of the file in total bytes, by setting the BX:CX registers to the number of bytes. Be careful that you have the BX set to zero before you write a COM file to the disk. Each digit in BX represents 65,000 decimal bytes. For our purposes BX will be always set to 0.